Inversion of Control (IoC) & *Dependency Injection* (DI) – 1

- *Inversion of Control*, The Secret Sauce of Frameworks
  - UI Example: Hardcoded Message Loop vs Event-Based Framework
  - The *Hollywood Principle*
  - Common IoC Patterns: *Observer*/*Slots & Signals*, *Template Method* (Hook)
- *Service Locator* Pattern
- How is Dependency Injection Different?
- Manual DI. DI Styles: Constructor > Field > Setter
- DI Standard (JSR 330): `javax.inject.`{@Inject,@Qualifier,@Named, Provider<T>}

UI: Hardcoded Message Loop


```
Dialog dlg = <...>;
Message msg;
while ((msg = dlg.pollMsg()) != null) {
    switch (msg.type()) {
        case CLICK:
            UI.toast("You're doomed!");
            break;
        default:
            dlg.defaultAction(msg);
    }
}
// dialog closes here
```

UI: Event-Based Framework

Callbacks customize framework actions

```
Dialog dlg = <...>;
dlg.addClickHandler(
    __ → UI.toast("Wheee!"));

// UI framework-managed msg
// loop starts here
int res = dlg.showModal();

// dialog closes here
```

Framework defines **Control** Flow

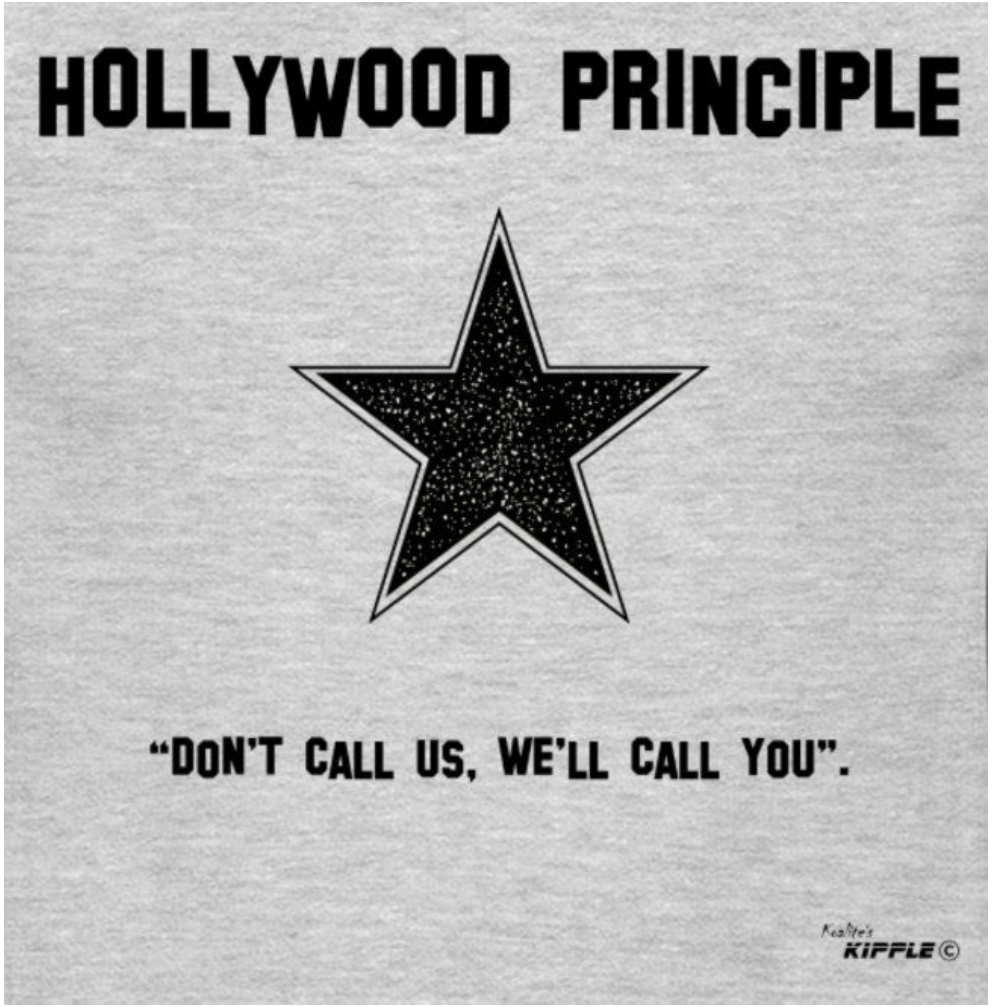
UI: Event-Based Framework (2)

```
public class Dialog extends <...> {
    public void addClickHandler(ClickHandler h) {
        dispatcher.add(h);
    }
    // <...>
    public void showModal() {
        Message msg;
        while ((msg = dlg.pollMsg()) != null) {
            dispatcher.handlersOf(msg.type())
                .ifPresent(EventHandler::on)
                .orElse(this::defaultAction);
        }
    }
}
```

Callback Interface: Event Handler

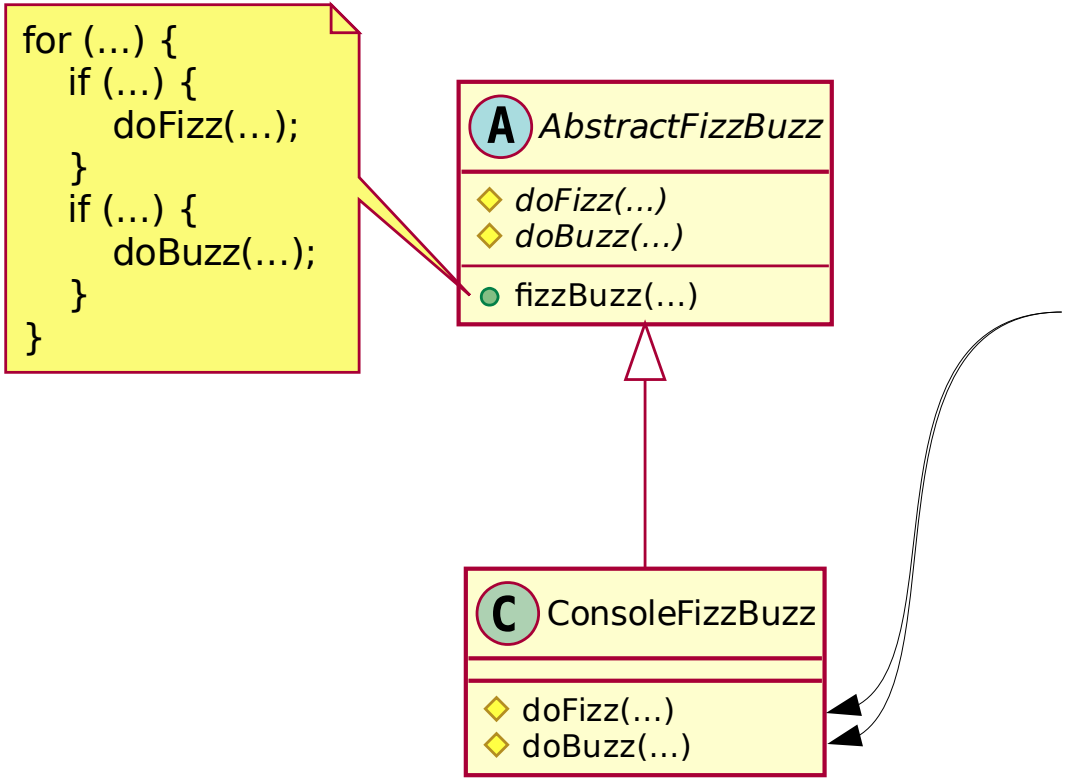
```
@FunctionalInterface
public interface ClickHandler
    extends EventHandler {

    void onClick(ClickEvent e);

    default void on(Message m) {
        this.onClick(<...>);
    }
}
```

UI Framework calls user code
@ specific lifecycle points
of Dialog's Message Loop



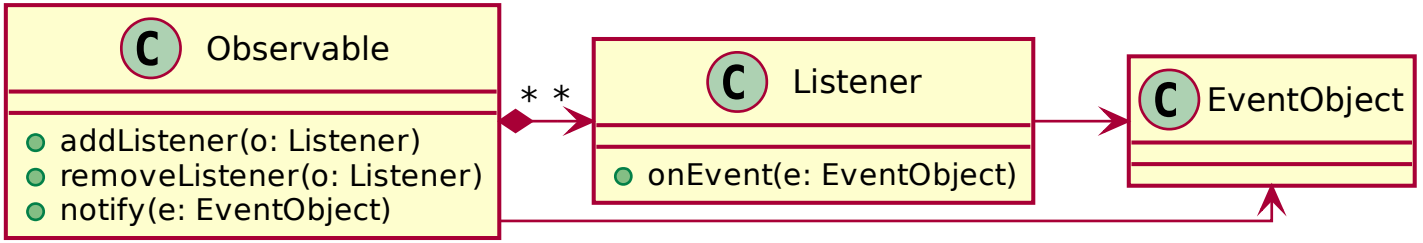
© Koalite's Kipple

IoC Pattern: Template Method



`doFizz()`, `doBuzz()` are **hooks** called by `fizzBuzz()` **@ specific lifecycle points**

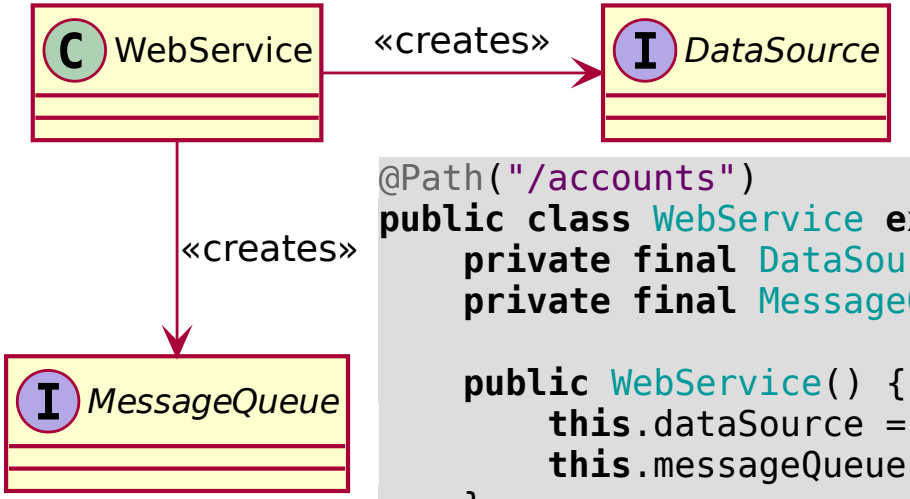
IoC Patterns: Observer



See Also: Signals & Slots (inspired by Qt)

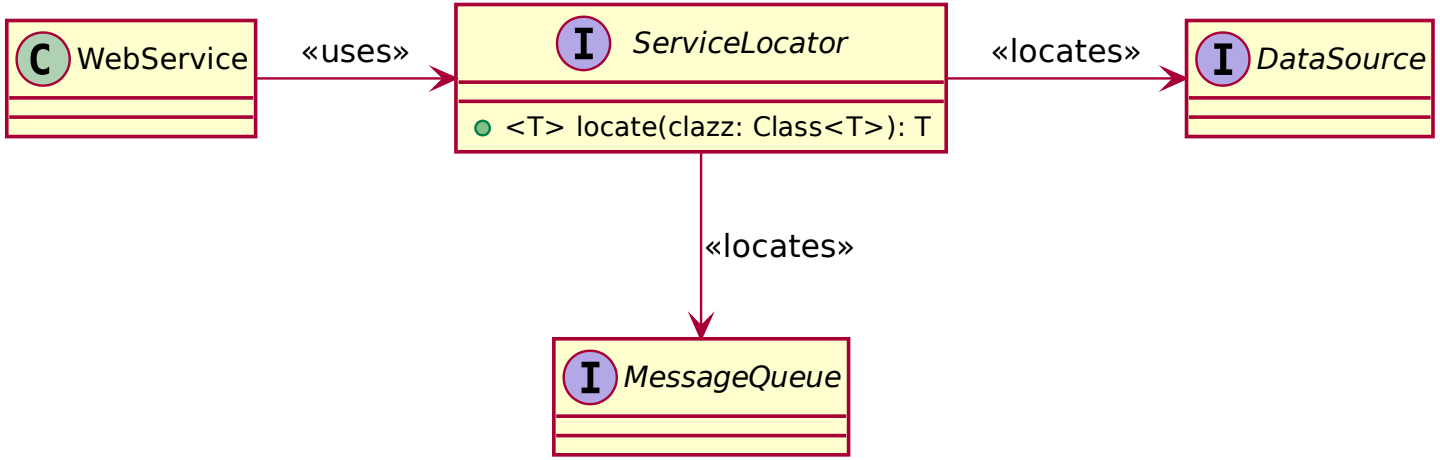
```
public class MyUIApp extends UIApplication<MyUIApp> {
    private final Signal1<EventObject> signal = new Signal1<>(...);
    public static void main(String... args) {
        MyUIApp app = new MyUIApp(args);
        app.signal.connect(eo → { // <...> });
        app.run();
    }
}
// Somewhere in framework UIApplication code, we emit the event:
this.signals(EventObject.class).emit(new EventObject(...));
```

Hardcoded Service Dependencies




```java
@Path("/accounts")
public class WebService extends <...> {
    private final DataSource dataSource;
    private final MessageQueue messageQueue;

    public WebService() {
        this.dataSource = new PgSqlDataSource("pgsql.infra", "...");
        this.messageQueue = new RabbitMessageQueue("rabbit.infra", 3232);
    }

    @POST
    @Consumes("application/json")
    public void createAccount(Account account) {
        var savedAccount = dataSource.persist(account);
        messageQueue.enqueue(new AccountCreatedEvent(savedAccount.id());
    }
}
```

Service Locator/Service Registry




```
public class WebService extends <...> {
    // <...>
    public WebService(ServiceLocator locator) {
        this.dataSource = locator.locate(DataSource.class);
        this.messageQueue = locator.locate(MessageQueue.class);
    }
    // <...>
}
```

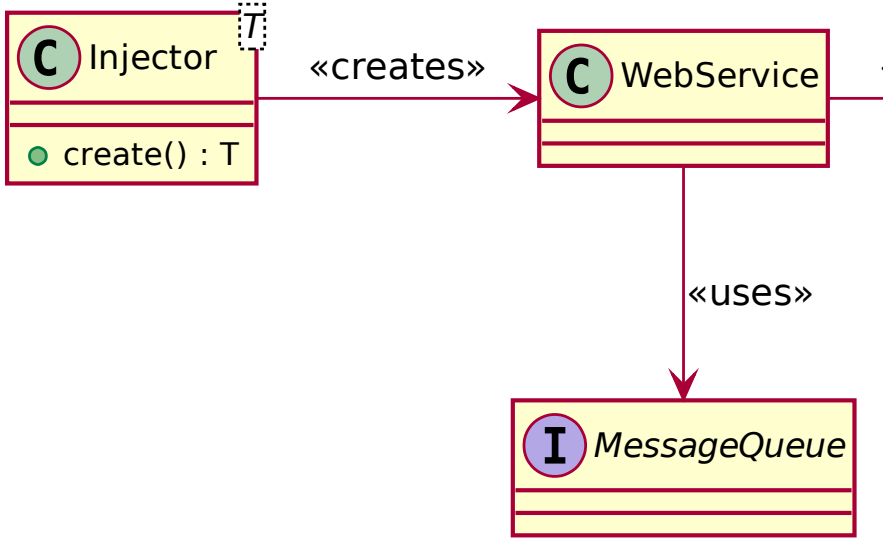
Service Locator is Fragile

```java
public class PgSqlDataSource implements DataSource {
    private final ServiceLocator locator;

    public PgSqlDataSource(ServiceLocator locator) {
        this.locator = locator;
    }

    public <T extends Entity<T>> persist(T entity) {
        PreparedStatement st = <...>;
        try (Connection conn = getConnection()) {
            return persisted(conn.executePrepared(st));
        } finally {
            // Hi, NPE!
            locator.locate(Tracer.class).traceDbQuery(st);
            // Is silently losing traces any better?
            locator
                .locateOrDefault(Tracer.class, __ → {})
                .traceDbQuery(st);
        }
    // <...>
}
```

```java
public class MR implements ServiceLocator {
  private final Map<Class, Object> m;
  public void <T> put(Class<T> c, T o) {
    if (null != m.putIfAbsent(c, o))
        throw <...>
  }

  public <T> locate(Class<T> clazz) {
    return clazz.cast(m.get(clazz));
  }
}
```

```java
public class FMR implements ServiceLocator {
  private final Map<Class, Object> m;
  private final Map<Class, Supplier> sm;
  public void <T> put(Class<T> c,
        Function<ServiceLocator, T> s) {
    if (null != sm.putIfAbsent(c, sm))
        throw <...>
  }

  public <T> locate(Class<T> clazz) {
    return m.computeIfAbsent(clazz,
        c → sm.get(clazz).apply(this));
  }
}
```

Dependency Injection




```
public class WebService extends <...> {
    public static void main(String... args) {
        Injector.root(WebService.class)
            .configure(<...>)
            .create().run(args);
    }
    @Inject
    public WebService(DataSource ds,
                      MessageQueue mq) {
        this.dataSource = ds;
        this.messageQueue = mq;
    }
    // <...>
}
```

Manual DI

```java
public class WebService extends <...> {
    public static void main(String... args) {
        new WebService(
            dataSource(),
            messageQueue().get()
        ).run(args);
    }

    public WebService(DataSource ds,
                      MessageQueue mq) {
        this.dataSource = ds;
        this.messageQueue = mq;
    }
    // <...>
```

```java
private static DataSource dataSource() {
    return new PgSqlDataSource(
        connectionPool(),
        <...>
    );
}

private static Supplier<MessageQueue>
    messageQueue() {
    return Suppliers.memoize(() →
        new RabbitMessageQueue(<...>));
}

// <...>
```

Standard DI: JSR 330 (`javax.inject`)

- `@Inject`: Declare dependency interfaces
  - constructors > fields > setters
- `@Qualifier`: Inject different implementations of the same interface
  - `@Named`: Choose impl by (unique) name
- `Provider<T>`: Lazy and optional injection
  - Essentially the same as `Supplier<T>`

Recommended Reading

- Martin Fowler, IoC Concepts: https://martinfowler.com/bliki/InversionOfControl.html
- Martin Fowler, DI (w/lots of code!): https://martinfowler.com/articles/injection.html
- DI vs Service Locator: https://sergeyteplyakov.blogspot.com/2013/03/di-service-locator.html